UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Aplicación de detección de ritmo cardíaco mediante análisis de secuencias de vídeo

Sergio García Ruiz

Tutor: José María Martínez Sánchez.

Junio 2017

Aplicación de detección de ritmo cardíaco mediante análisis de secuencias de vídeo

Sergio García Ruiz
Tutor: José María Martínez Sánchez.

Video Processing and Understanding Lab
Departamento de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2017

Trabajo parcialmente financiado por el Ministerio de Economía y Competitividad del Gobierno de España bajo el proyecto TEC2014-53176-R (HAVideo) (2015-2017)

Resumen

El objetivo de este trabajo de fin de grado es crear una aplicación que integrase tanto un algoritmo de detección de ritmo cardíaco basado en secuencias de vídeo (en modalidad color y profundidad), una conexión con un elemento que permitiese medir el ritmo cardíaco real de un sujeto y todo ello centralizado en una aplicación que mostrase los resultados de ambos en una interfaz.

Se ha diseñado una aplicación basándonos en las necesidades establecidas. Desarrollada en C# como lenguaje principal para poder hacer uso de las funcionalidades del Kinect V2, elemento sobre el que gira la aplicación. Con el fin de adquirir los valores reales del ritmo cardíaco se ha elaborado un sistema basado en el protocolo Bluetooth empleando las funcionalidades proporcionadas por la paquetería de Ubuntu siendo esta la mejor opción tras evaluar las alternativas. Para el algoritmo de estimación del ritmo cardíaco se ha implementado una versión en C# utilizando como base algoritmo funcionales desarrollados en el MIT, modificando dicho algoritmo con el fin de conseguir el rendimiento necesario.

Todas estas componentes se han integrado en una aplicación MVC sobre Visual Studio con el fin de facilitar y agilizar el desarrollo, además de mostrar los resultados de ambas aproximaciones a la medición del ritmo cardíaco, la aplicación almacenará los datos para su posterior uso en el desarrollo y evaluación de nuevos algoritmos de estimación del ritmo cardíaco a partir de secuencias de vídeo de nuevas versiones del algoritmo de detección implementado.

Finalmente se han comprobado el funcionamiento del algoritmo con una tanda de mediciones empleando las funcionalidades presentes en la aplicación desarrollada.

Palabras clave

Ritmo cardíaco, Kinect, análisis de secuencias de vídeo, Modelo Vista Controlador, C#, Visual Studio, Javascript, puntos de interés, Shell, algoritmo, Transformada de Fourier.

Abstract

The objective of this final degree thesis is to create an application that integrates a heart rate detection algorithm based on video sequences (color and depth mode), a connection with an element that allows us to measure the real heart rate of a subject and everything centralized in an application that display the results of both elements in an interface.

The application developed has been designed based on the established needs. Developed in C# as main language to be able to use the main features of Kinect V2, core element of the application. In order to acquire the real values of heart rate, a system based on Bluetooth protocol has been developed using tools and functions of the Ubuntu packaging being this the best option after the analysis of the alternatives. For the heart rate estimation algorithm, we have implemented a C# version based on an algorithm developed in the MIT, modifying that algorithm in order to achieve the required performance.

All parts have been integrated into an MVC application on Visual Studio in order to make development easier and faster, in addition to showing the results of both approaches to heart rate measurement, the application store all data for later use in development and evaluation of new heart rate estimation algorithms implemented.

Keywords

Kinect, video sequence analysis, Model View Controller, C#, Visual Studio, Javascript, points of interest, shell, algorithm, Fourier Transform.

Agradecimientos

Quiero dar las gracias a mi tutor Chema por darme apoyo y ayudarme a realizar este TFG a pesar de todas las dificultades que he encontrado por el camino.

Además quiero dar las gracias a mis compañeros de carrera que han hecho más llevadero todos estos años David, Néstor, Galán, Javi, Alfonso y Gus.

Gracias también a mi gran amigo Carlos que compartió conmigo tantas y tantas prácticas y que conseguía sacarme una carcajada incluso en los momentos más duros.

No me olvido por supuesto de toda mi familia que me dado todo el apoyo del mundo durante toda la carrera y en esos momentos cuando más lo necesitaba y siempre me han dado ánimos para que no tirase la toalla.

Finalmente quiero dar las gracias a Dayvid que me dio confianza hasta el último día d.e.p compañero nunca te olvidaremos.

Sergio García Ruiz

Junio 2017

Índice general

Índice de figuras

Índice de tablas

1. Introducción

1.1 Motivación

El ritmo cardíaco es uno de nuestros signos vitales, se define como el número de veces por minuto que nuestro corazón se contrae. El uso de este elemento vital ha desarrollado mucha importancia en los últimos años y su uso ya no se limita solo al mundo del deporte. Relojes, pulseras de actividad incluyen sensores que permiten monitorizar el estado el estado de esta métrica en busca de posibles anomalías.

A raíz de esto, se desarrollaron nuevas medidas para el cálculo de estos ritmos intentando reducir el contacto con el usuario e incluso realizándose a distancia. Se han desarrollado algoritmos que permiten estimar el ritmo cardíaco analizando valores como el cambio de color en la cara fruto del bombeo de la sangre o como el movimiento que genera en la cabeza el flujo sanguíneo.

Este trabajo propone implementar un algoritmo que permita estimar el ritmo cardíaco mediante análisis de secuencias de vídeo, donde se tratara de analizar el movimiento cefálico inducido por los movimientos que la sangre produce al alcanzar la cabeza. Además de esto se pretende desarrollar una aplicación que permita analizar la precisión del algoritmo en tiempo real comparando los resultados con los valores obtenidos de un pulsímetro real, todo ello centralizado en una aplicación de escritorio. Finalmente, la aplicación permitirá grabar las imágenes en color y profundidad y los resultados reales del pulsímetro con el fin de poder usar estos datos en el desarrollo y evaluación de nuevos algoritmos de estimación del ritmo cardíaco a partir de secuencias de vídeo.

1.2 Objetivos

Para desarrollar este trabajo se definen los siguientes objetivos:

- Estudio del estado del arte.
- Estudio del funcionamiento de C# y Kinect V2.
- Desarrollo del módulo Kinect.
- Integración del módulo Kinect con aplicación MVC.

- Estudio del funcionamiento de BLE y PolarH7.
- Integración del módulo pulsímetro con la aplicación MVC.
- Desarrollo de algoritmo de detección de ritmo cardíaco en C#.
- Integración del algoritmo con la aplicación.
- Mediciones y análisis final del funcionamiento del algoritmo.

1.3 Organización de la memoria

La memoria se compone de las siguientes secciones:

- **Capítulo 1**: Motivación, objetivos y distribución de la memoria
- **Capítulo 2**: Estado del arte en el análisis del ritmo cardíaco y su obtención mediante análisis de secuencias de vídeo.
- **Capítulo 3**: Diseño de la aplicación y de sus componentes principales.
- **Capítulo 4**: Desarrollo de las principales componentes de cada uno de los componentes de la aplicación.
- **Capítulo 5**: Descripción de los resultados del algoritmo desarrollado.
- **Capítulo 6**: Conclusiones y trabajo futuro.
- Referencias
- Apéndice A: Manual de Instalación
- Apéndice B: Kinect Face y Kinect Face HD

2. Estado del arte

2.1 Introducción

El estudio del ritmo cardíaco, así como las técnicas de obtención del mismo han experimentado un increíble aumento en los últimos años, el seguimiento de este valor permite detectar problema de salud o prever diversas patologías. El número de dispositivos que incluyen sensores para medir estos valores es cada vez mayor incorporándose actualmente en smartphones, smartwatches o pulseras de actividad entre otros. Por ese motivo se están desarrollado métodos menos intrusivos que eliminan la necesidad de tener un contacto directo con el usuario y la detección del ritmo cardíaco mediante análisis de secuencias de vídeo permite detectar el ritmo cardíaco de un individuo a distancia.

Existen dos métodos predominantes en este campo: detección mediante variaciones de color y mediante movimiento.

Los métodos basados en variaciones en el color emplean los cambios en la tonalidad de la piel (por ejemplo, de la cara o de un dedo) provocados por la circulación de la sangre y los métodos basados en movimiento aprovechan los movimientos que produce la sangre en la cabeza al avanzar por la aorta. Tanto los cambios en el color como los movimientos cefálicos resultan imperceptibles para el ojo humano, pero gracias a técnicas de amplificación de vídeo desarrolladas en el MIT [1], se han hecho visibles y permiten un análisis más preciso de estas características.

2.2 Obtención del ritmo cardíaco mediante análisis de vídeo

2.2.1 Métodos basados en cambios de color

Este método es el más empleado, se analiza los cambios de color ocasionados en la piel, generalmente de la cara o de un dedo, por la circulación de la sangre. Durante la sístole o fase de contracción la piel tiene una tonalidad más rojiza mientras que en el periodo de diástole o dilatación la piel adquiere un tono más pálido o amarillento. Estas

variaciones se analizan y permiten realizar una estimación del ritmo cardíaco del sujeto calculando la frecuencia a la que tenemos la máxima energía en un espacio transformado. Actualmente existen algunas aplicaciones en el mercado que emplean esta idea como Cardiio [2] o Heart Rate Monitor [3].

Cardiio App. Fuente: [27] Heart Rate Monitor App. Fuente: [28]

Figura 2.1: Aplicaciones basadas en análisis de cambios de color.

Estas aplicaciones realizan una detección de la cara empleando algoritmos como Viola-Jones [2] o mediante la transformada de Haar entrenada para detectar determinados patrones de iluminación en el rostro.

En el estado del arte se pueden encontrar diversos métodos para analizar los cambios de color se emplean los siguientes métodos:

- Comparar *frames* en HSI [3]
- Separar la imagen en R, G y B y normalizar para aplicar Análisis de Componentes Independientes (ICA, *Independent Components Analysis*) o Análisis de Componentes Principales (PCA, *Principal Components Analysis*) [4].
- Separar la imagen en R, G y B y realizar un postprocesado sin utilizar ICA o PCA [5].
- Realizar la operación AND de la imagen binarizada mediante la clasificación del tono de piel con el uso de la escala cromática de Fitzpatrick [6] y la imagen en el espacio YUV [7].
- Mediante el cálculo de las diferencias de brillo [8].

- Uso de descomposición piramidal y análisis de la componente R [9].

También tenemos distintas variantes a la hora de obtener la frecuencia dominante, como, por ejemplo, mediante el cálculo de la Transformada Rápida de Fourier (FFT - *Fast Fourier Transform*) [10] o el uso de la Densidad Espectral de Potencia (PSD - *Power Spectral Density*) [11].

2.2.1 Métodos basados en movimiento

En 2013 se realizó el primer trabajo que empleaba los movimientos cefálicos producidos por la circulación de la sangre en la cabeza por parte de Balakrishnan et al. [12] en el MIT (*Massachusetts Institute of Technology*). En primer lugar, se realizaba una detección de la cara mediante el algoritmo de Viola-Jones, y se localizaban los principales puntos de interés (POI, *Points of Interest*) contenidos en la zona detectada mediante el algoritmo *Good Features to Track* [13]. Para realizar el seguimiento de los POI en cada una de las imágenes de la secuencia se emplea el algoritmo KLT (*Kanade Lucas Tracking*) [14], que permite obtener la trayectoria de cada uno de los puntos detectados. Finalmente se emplea la Transformada de Fourier sobre las trayectorias filtradas para la obtención del ritmo cardíaco.

En 2014 en la Universidad de Aalborg, Irani et al [15] se realizaron algunas modificaciones sobre el algoritmo del MIT, como son el uso de un filtro de suavizado después del filtrado de la señal o el uso de la Transformada Discreta del Coseno en lugar de la DFT.

En 2015 Erik Velasco Salido en su Trabajo de Fin de Grado [16], utilizó el algoritmo propuesto por el MIT y le aplicó mejoras para poder utilizar las secuencias de imagen en profundidad proporcionadas por la Kinect.

En 2016 Guillermo Guillen en su trabajo de Fin de Carrera [17], realizó mejoras en la captura de imágenes empleando una Kinect 2.0 además de estudiar como afectaba al resultado del algoritmo la compresión de las imágenes de salida, y analizando como afectaba el uso de la frecuencia media para mejorar la estimación del ritmo cardíaco.

2.4 Conclusiones

Basándome en las observaciones anteriores se ha decidido basar este trabajo en la rama de detección del ritmo cardíaco basado en el análisis del movimiento en secuencias de vídeo, ya que a diferencia del otro modelo permite emplear esta técnica no solo con imágenes de color (RGB) sino también con imágenes de profundidad o imágenes infrarrojas todas ellas proporcionadas por la Kinect V2.

3. Diseño

3.1 Diseño general de la aplicación

Para el diseño de esta aplicación se ha usado el patrón de diseño Modelo-Vista-Controlador [25].

El MVC es un patrón de diseño software en el que se separa la lógica interna de la aplicación, la interfaz gráfica de usuario y la comunicación entre estos dos elementos se coordina a través del controlador. Una de las mayores ventajas de este modelo es la abstracción de los elementos, nunca se debe hacer referencia desde el modelo a un objeto del navegador y viceversa, de esta manera la repercusión producida por una modificación en alguno de los agentes es mínima y no se propaga a toda la aplicación.

En ese caso se ha elegido este patrón por la naturaleza del problema. Cada uno de los módulos posee una lógica interna muy compleja pero la vista no emplea más que el resultado de toda esa lógica, lo que permite la manipulación mucho más segura de estos módulos.

Los módulos contenidos en el modelo no intercambian información en forma de código, no comparten variables, métodos o clases entre ellos y solo se comunican mediante ficheros, para evitar problemáticas de efecto colateral (*Side Effect*) como se podrá ver en la siguiente sección.

El modelo se compone de 3 módulos:

- Módulo Kinect: encargado de realizar las acciones con la Kinect 2.0 y generar las imagines para la vista
- Módulo Pulsímetro: encargado de comunicarse con la banda Polar y obtener las pulsaciones reales del sujeto.
- Módulo Algoritmo: encargado de operar con los datos de la Kinect y obtener las estimaciones del ritmo cardíaco.

A parte del patrón de diseño empleado, la aplicación se comporta como una API REST. Al desplegarla se expone un puerto y mediante parámetros en la URL se modifica el

funcionamiento de la aplicación. Desde el navegador todas las peticiones y parámetros son encapsulados y se realizan a través de los distintos botones de la vista, y los parámetros son recibidos en el controlador donde se analizan y se *pasean* antes de generar las operaciones en el modelo.

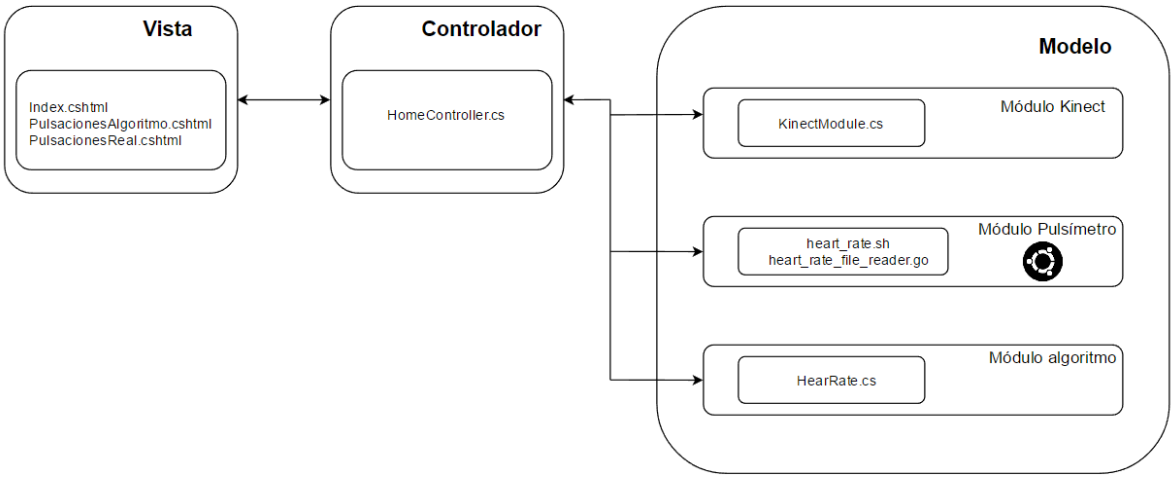

Figura 3.1: Diagrama general de la aplicación

3.1.1 Módulo Kinect

El módulo Kinect es el encargado de realizar las comunicaciones con la API v2 de Windows para Kinect [18]. La lógica de este módulo se encuentra desarrollada en la clase **KinectModule** contenida en el fichero KinectModule.cs. Se encarga del manejo de sesiones de instancias del sensor de la Kinect, de manejar la apertura y cierre del mismo, y de transformar los objetos de salida que proporciona la API como son *MultiSourceFrame* en elementos que se pueden mostrar en navegadores convencionales.

Uno de los requisitos de este módulo era que debía ser muy eficiente, además de ser capaz de realizar tareas de manera concurrente, ya que cuanto mayor sea número de *frames* procesados por segundo mejor será el resultado del algoritmo. Finalmente se alcanzaron los 27 *frames*/segundos de manera estable en situaciones de abundante luz.

3.1.2 Módulo Pulsímetro

Este módulo es el encargado de realizar la comunicación con la banda Polar H7 [19], se encarga de realizar las configuraciones del dispositivo, así como de *pasear* la información de salida del sensor. Se compone de un script en bash, **heart_rate.sh**, que automatiza la comunicación inicial que es necesaria para sincronizarse con el dispositivo, y de un programa en Golang que analiza y depura la salida de este script y la transmite a la aplicación. Este módulo debe ejecutarse sobre un ordenador con Ubuntu ya que emplea paquetes que no se pueden emplear en la plataforma de Microsoft.

Para eliminar esta dependencia de un segundo sistema operativo se estudió la posibilidad de utilizar las Windows UWP (Universal Windows Platform). Con la creación de Windows 8, se introdujo el WinRT (Windows Runtime) con el fin de facilitar el desarrollo de aplicaciones para la plataforma de Microsoft. Con la salida de Windows 10 se creó la Plataforma Universal de Windows, con la idea de crear una única manera de desarrollo compartido entre aplicaciones para Pc, Móvil, Xbox, IoT (*Internet of things*) etc. Una de las características que proporciona UWP es la conexión con dispositivos Bluetooth de Baja Energía o BLE como la banda Polar H7. La clase **HeartRate.cs** contiene una prueba de concepto de esta tecnología, sin embargo, se acabó descartando por varios motivos.

En primer lugar, antes comenzar la comunicación con el dispositivo este debe estar sincronizado previamente con el ordenador que tenga la aplicación instalada ya que si no es así las llamadas a la API de UWP no detectarán el dispositivo y no se podrá realizar la sincronización. Y, en segundo lugar, y lo más condicionante, es que en la versión actual Visual Studio 2015 produce una saturación y no se pueden acceder a más de 10 mediciones sin tener que reiniciar este módulo.

3.1.3 Módulo Algoritmo

Este módulo es el encargado de aplicar el algoritmo de detección de ritmo cardíaco mediante imágenes de Kinect basado en el desarrollado por Erik Velasco [16]. Sin embargo, este algoritmo estaba desarrollado íntegramente en Matlab. Este lenguaje a diferencia de otros como Golang no es OpenSource y el contenido de sus funciones y su lógica interna no se puede consultar. Por esto último se estudiaron medidas para trasladar el funcionamiento de la aplicación a C# con la mayor precisión posible. Una de las características que proporciona C# es el uso de funciones Matlab embebidas en su

código, aunque se descartó esta opción debido a la lentitud de estas llamadas. Algunas de las operaciones más frecuentes como la Transformada de Fourier, o el uso de la función *CascadeObjectDetector* para la detección de caras en imágenes ralentizaban demasiado el desarrollo del programa y eliminaban la posibilidad de dar resultados en periodos inferiores a 5 segundos.

Uno de los mecanismos estudiados para integrar el algoritmo propuesto con la versión Kinect fue hacer uso de las características de *Kinect Face* (ver Apéndice B). A principios de 2015 se añadió una serie de funcionalidades al SDK de Kinect que permitían hacer el seguimiento de una serie de puntos característicos de la cara, y que permitían entre otras cosas detectar movimientos o rotaciones en la cabeza de los usuarios. Para reducir los tiempos empleados por el algoritmo en detección y seguimiento de puntos se ha hecho uso de estas funciones.

3.1.4 Controlador

El controlador contenido en la clase **HomeController.cs** es el encargado de comunicar la vista con el modelo y viceversa. Es el lugar en el que se procesan los parámetros contenidos en las peticiones POST procedentes del navegador. Los parámetros que acepta el controlador son los siguientes:

- *mode*: modifica el tipo de imagen mostrada en la vista. 0 muestra imágenes en color y 1 imágenes de Profundidad.
- *frameRate*: permite modificar la tasa de captura de las imágenes generadas por el kinect.
- *recordTime*: en caso de realizar una grabación, este valor contiene los segundos que se grabarán.

Estos parámetros se procesan y se realizan las tareas necesarias en los modelos, se generan las vistas y se envían al navegador que realizó la petición.

El controlador también se encarga de gestionar todas las tareas que implican realizar operaciones con elementos de los módulos, como son:

- Lectura de fichero compartido Ubuntu-Windows con el contenido de las pulsaciones reales.
- Lectura de fichero resultado del módulo pulsímetro con el ultimo resultado estimado por al algoritmo.
- Gestión de la tasa de refresco de las imágenes proporcionadas por el Kinect.
- Persistencia a disco de *frames* en color, profundidad, pulsaciones reales y pulsaciones estimadas de un intervalo de tiempo.
- Creación de evento de escucha de *frames* generados por el Kinect.

3.1.5 Vista

Las distintas vistas se encuentran en los ficheros Index.cshtml, PulsacionesAlgoritmo.cshtml y PulsacionesReal.cshtml.

Se estudiaron varias tecnologías a la hora de mostrar los *frames* extraídos del Kinect en un navegador. La primera seria utilizar las tecnologías proporcionadas por el sistema de aplicaciones de Windows. XAML (*eXtensible Application Markup Language*) es un lenguaje de marcado desarrollado por Microsoft y forma parte de Microsoft Windows Presentation Foundation (WPF). WPF es la categoría del *framework* .NET relacionada con la presentación visual de aplicaciones basadas en Windows, y tanto XAML como WPF se emplean con la idea de agilizar la escritura de aplicaciones en C#.

Al formar parte del conjunto de desarrollo de Microsoft XAML permite mostrar las imágenes del Kinect en el navegador de una manera sencilla, ya que puede recibir directamente el *frame* en color o profundidad convertido en **Bitmap**. Sin embargo, se intentó eliminar las dependencias de este tipo con la idea de abstraer lo máximo posible la vista de la plataforma de Windows, y se desarrolló una serie de funciones que permiten trabajar con estos *frames* directamente en un navegador empleando HTML&CSS y Javascript.

Para la comunicación entre Controlador y Vista se ha empleado Razor (ficheros con terminación cshtml). Razor es un lenguaje que permite utilizar código C# embebido en HTML, y se ha empleado para transmitir *arrays* de *strings* desde el controlador a la vista con infamación como las dimensiones de las imágenes mostradas que varía en función del modo seleccionado.

4. Desarrollo

En esta sección se comentará más en profundad cómo se han implementado cada uno de los elementos de la aplicación.

4.1 Módulo Kinect

Este módulo es el encargado de realizar la comunicación con el sensor Kinect para la obtención de las imágenes capturadas por el dispositivo. La comunicación se realiza directamente contra la API de *Kinect for Windows V2* [18].

La función principal de este módulo es `KinectFrameArrivedEvent`. Esta función se ejecuta cada vez que se produce un evento del tipo `MultiSourceFrameArrived`.

Un evento en C# es el modo que tiene una clase u objeto de realizar notificaciones a los clientes de la clase cuando se produce una determinada modificación en el objeto. De esta manera se elimina la necesidad de estar comprobando constantemente un atributo o una propiedad, además son especialmente útiles en el caso de sensores o elementos que realizan modificaciones en sus propiedades de manera asíncrona.

Y de esta manera se realiza la suscripción del evento:

```
kinect.Reader.MultiSourceFrameArrived += kinect.KinectFrameArrivedEvent;
```

La función mencionada, `KinectFrameArrivedEvent`, desencadena el flujo que se realiza sobre cada uno de los *frames* producidos en el Kinect.

Al tratarse de una aplicación de tipo web cada petición POST realizada por el navegador se genera una nueva instancia de la clase **HomeController** y un nuevo **KinectSensor**. Para evitar problemas relacionados con mantener varios **KinectSensor** se ha creado un mecanismo similar al patrón Singleton en el método *lockKinectSesion*.

Al llamar al método que instancia el **KinectSensor** se le proporciona un id entero generado en el controlador de manera aleatoria, que se pasa como parámetro al crear el objeto y con el que se genera un fichero *.lock* que almacena dicho id. Cada n *frames*, 30 actualmente, se comprueba el estado del fichero con el fin de comprobar si el id ha

sido modificado. Si al leer el id del fichero este no concuerda con el del objeto significa que se ha creado una nueva instancia del sensor y por lo tanto la instancia que está comprobando el fichero *.lock* debe ser eliminada, eliminando antes las suscripciones a los eventos del Kinect.

Tras la realización de las operaciones de control de la sesión se abre el sensor de la Kinect donde se debe indicar los canales que se quieren recibir cada vez que se reciba un *frame* (en este caso color y profundidad). Cada vez que genere un *frame* en el Kinect se recibirá una estructura, **MultiSourceFrame**, que contiene el *frame* en todos los formatos seleccionados. Es importante la selección de los canales a la hora de abrir el sensor, ya que cuantos más canales se seleccionen más grande será cada instancia de **MultiSourceFrame** y dada la alta tasa de recepción esto puede derivar en serios problemas de rendimiento.

Finalmente se realiza las transformaciones del *frame* recibido con el fin de adaptarlo a un formato fácil de manejar. En este caso se transforma en formato **Bitmap** y dicho **Bitmap** se escribe a disco para ser leído por la vista y mostrarlo en el navegador, además de para la creación de bases de datos para futuro uso de desarrollo y evaluación de algoritmos de estimación de ritmo cardíaco.

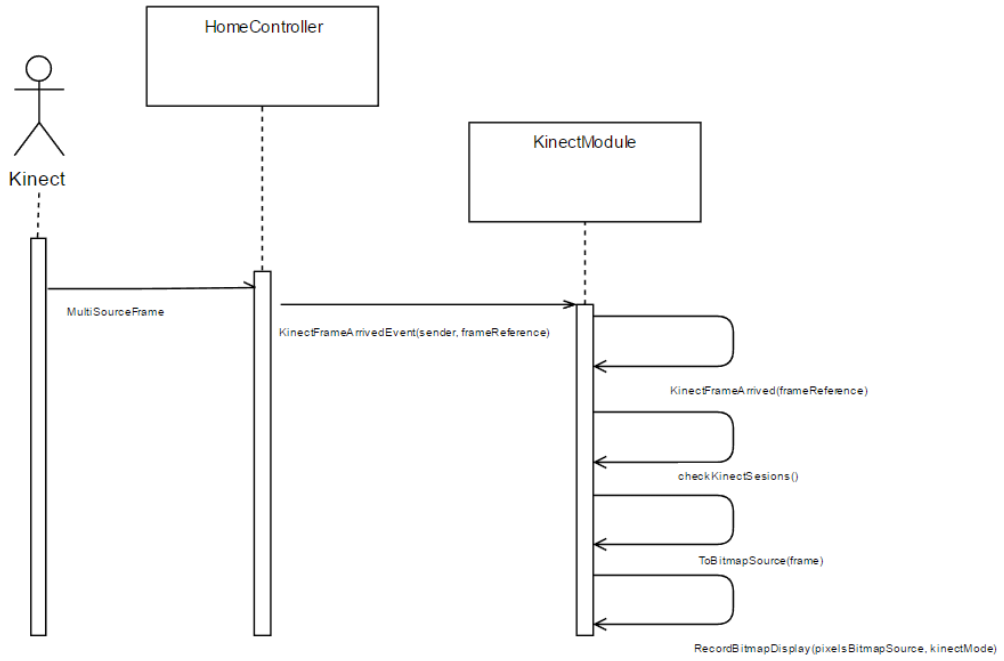

Figura 4.1: Flujo de manejo de un MultiSourceFrame

En la Figura 4.1 se puede ver las operaciones realizadas sobre los **MultiSouceFrame** recibidos para mostrarlos en la vista, este proceso es muy similar al que se realiza con los *frames* para almacenarlos en caso de pulsar la opción Record en la interfaz.

4.2 Módulo Pulsímetro

4.2.1 Script de obtención del Ritmo Cardíaco

Este módulo, como se explicó en el diseño se debe ejecutar sobre un ordenador con Ubuntu como Sistema Operativo. Para realizar la comunicación con un dispositivo Bluetooth de Baja Energía (BLE) se ha empleado un paquete que se encuentra instalado en las últimas distribuciones de Ubuntu, y que contiene los siguientes comandos que vamos a emplear: hcitool y gatttool.

En primer lugar, es necesario obtener la interfaz Bluetooth del dispositivo mediante hcitool, ya que el nombre de esta interfaz varía en distintos dispositivos y es necesario conocerla para la comunicación. Con el nombre de la interfaz se pude realizar un barrido para identificar el nombre y MAC de la banda de la que obtendremos las pulsaciones. Es importante indicar que algunos dispositivos BLE como la banda Polar H7 no pueden atender peticiones simultáneas de varias fuentes [24], y si está conectado con algún dispositivo como un *Smartphone* no contestara a los intentos de inicio conexión del script.

Con la información de la banda se puede iniciar la comunicación con el dispositivo. A continuación, se describe el proceso de acceso a los valores de manera manual, aunque en el script se ha realizado de una manera más rápida empleando los valores específicos de la banda empleada.

```
sergio@sergiogr:~$ sudo gatttool -i hci0 -b 00:22:D0:B4:19:E2 -I
[00:22:D0:B4:19:E2][LE]> connect
Attempting to connect to 00:22:D0:B4:19:E2
Connection successful
[00:22:D0:B4:19:E2][LE]> primary
attr handle: 0x0001, end grp handle: 0x000b uuid: 00001800-0000-1000-8000-00805f9b34fb
attr handle: 0x000c, end grp handle: 0x000f uuid: 00001801-0000-1000-8000-00805f9b34fb
attr handle: 0x0010, end grp handle: 0x0015 uuid: 0000180d-0000-1000-8000-00805f9b34fb
attr handle: 0x0016, end grp handle: 0x0024 uuid: 0000180a-0000-1000-8000-00805f9b34fb
attr handle: 0x0025, end grp handle: 0x0027 uuid: 0000180f-0000-1000-8000-00805f9b34fb
attr handle: 0x0028, end grp handle: 0xffff uuid: 6217ff4b-fb31-1140-ad5a-a45545d7ecf3
```

Figura 4.2 Conexión con dispositivo BLE

Mediante la documentación de ritmo cardíaco de bluetooth [22] conocemos que la componente del ritmo cardíaco es la que comienza con el uuid: 0x1800d, y necesitaremos los valores de los atributos attr handle y grp handle, ya que la

información de las pulsaciones se encuentra contenida en la sección limitada por los descriptores 0x0010 y 0x0015.

```
[00:22:D0:B4:19:E2][LE]> char-desc 0x0010 0x0015
handle: 0x0010, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x0011, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0012, uuid: 00002a37-0000-1000-8000-00805f9b34fb

handle: 0x002d, uuid: 00002902-0000-1000-8000-00805f9b34fb
handle: 0x0010, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x0011, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0012, uuid: 00002a37-0000-1000-8000-00805f9b34fb
handle: 0x0013, uuid: 00002902-0000-1000-8000-00805f9b34fb
handle: 0x0014, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0015, uuid: 00002a38-0000-1000-8000-00805f9b34fb
```

Figura 4.3: Obtención de descriptor del ritmo cardíaco.

Accedemos a los descriptores que contienen la información del ritmo cardíaco y finalmente hacemos una escritura en el manejador indicado en [23] para empezar a recibir los valores del ritmo cardíaco de la banda en valor hexadecimal.

```
[00:22:D0:B4:19:E2][LE]> char-write-req 0x0013 0100
Characteristic value was written successfully
Notification handle = 0x0012 value: 06 00
Notification handle = 0x0012 value: 06 00
Notification handle = 0x0012 value: 16 00 da 18 10 01
Notification handle = 0x0012 value: 16 00 17 05 3f 01
Notification handle = 0x0012 value: 16 00 d6 02 83 01
Notification handle = 0x0012 value: 16 00 29 01 26 01
Notification handle = 0x0012 value: 06 00
Notification handle = 0x0012 value: 16 00 80 08
Notification handle = 0x0012 value: 16 00 af 01
Notification handle = 0x0012 value: 06 00
```

Figura 4.4: Valores actuales del ritmo cardíaco.

4.2.2 Programa en Golang

Además del script se ha creado un programa en Golang que obtiene la salida del script anterior y devuelve un valor entero que es el escrito en la carpeta compartida entre Windows y Ubuntu y que recibe la aplicación. Dado que la escritura de los valores del sensor es bloqueante no se puede realizar este *parseo* en el propio script y es por esto que se decidió crear este programa heart_rate_file_reader.go.

4.3 Módulo Algoritmo

Este módulo contiene la lógica que implementa el algoritmo de detección de ritmo cardíaco mediante análisis de las imágenes de salida de la Kinect. El algoritmo en el que me he basado para este objetivo es la versión del algoritmo desarrollado por Balkrishnan en el MIT [12], así como algunas de las mejoras realizadas por Erik Velasco [16] sobre ese mismo algoritmo.

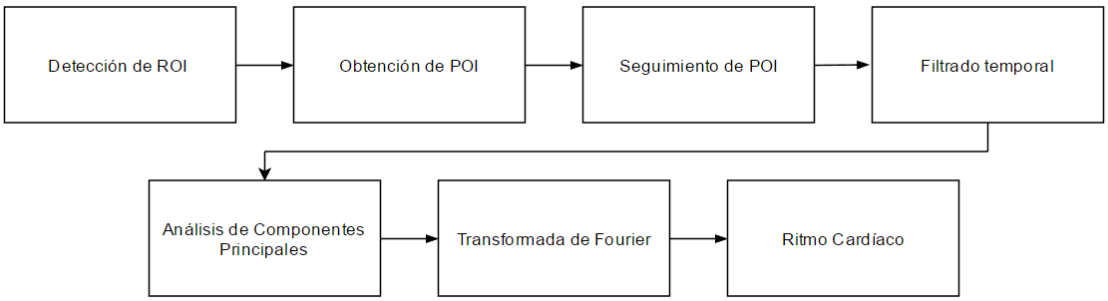

Figura 4.5: Funcionamiento Algoritmo de Balkrishnan

En primer lugar, se realiza una detección de las regiones de interés, en este caso estas regiones se encuentran localizadas en la cara del sujeto, para esta detección se empleaba el algoritmo Viola-Jones [2]. Dentro de esas regiones de interés se buscaban aquellos puntos que proporcionasen más información mediante el algoritmo *Good Features to Track* [13]. A continuación, se realizaba un rastreo de esos puntos de interés en los siguientes *frames* empleando para ello el algoritmo KLT [14] y se le realizaba un filtrado con el fin de eliminar ruidos en la señal producidos por movimientos involuntarios como parpadeos. Se realiza un análisis de componentes principales para reducir el número de muestras, tratando de perder el mínimo de información posible, y finalmente se aplicaba la transformada de Fourier sobre el PCA para detectar frecuencia del ritmo cardíaco contenido en el resultado del análisis de componentes.

Este algoritmo fue trasladado a Matlab por Erik Velasco y se realizó un análisis de su funcionamiento, así como una serie de mejoras. La idea de la aplicación a desarrollar era la de proporcionar este valor del ritmo cardíaco en tiempo real, y se pudiese verificar tanto el valor real como el estimado al mismo tiempo. Por este motivo se descartó la posibilidad de emplear Matlab embebido en C#, además realizar una traducción exacta del algoritmo también se descartó, ya que procesos como la detección de la cara del usuario tardaba 1 minuto mediante la transformada de Haar para la detección de nariz y frente. Para acelerar este proceso se decidió hacer uso de la característica de *Face tracking* disponible en la versión 2 de Kinect.

Para emplear las funciones de *Kinect Face* y acceder a los puntos característicos es necesario realizar unas acciones previamente descritas en el diagrama de la figura 4.6.

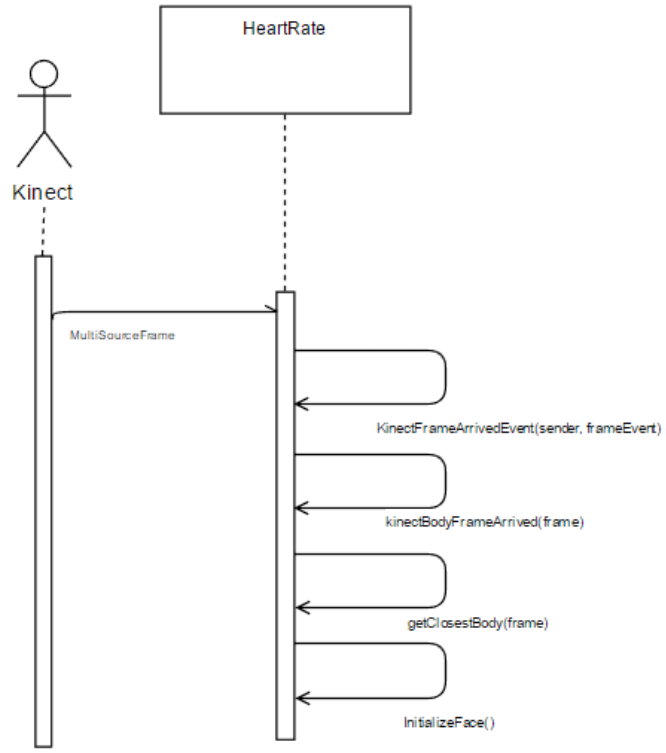

Figura 4.6: Flujo de obtención de Puntos de Interés

Al llegar un nuevo *frame* del Kinect se accederá al objeto **BodyFrame** contenido en el evento **MultiSourceFrameArrived** que contiene información sobre la posición de todas las personas en el ángulo de visión del Kinect. Cada persona reconocida por el Kinect tiene asociado un TrackingId que permite identificar y manejar la información de cada una de las personas en pantalla. En nuestro caso solo se almacena el estado de la persona más cercana a la cámara y solo se aplica el algoritmo sobre ésta. Una vez detectado el usuario objetivo se comienza a analizar la información de su rostro.

Empleando las características del *Face Tracking* podemos acceder a las coordenadas de una serie de puntos característicos como son: esquinas de ojo izquierdo y derecho, esquina derecha e izquierda de la boca y la punta de la nariz de los *frames* de color, y n cada *frame* devuelto de la Kinect se podrá consultar la posición de cada uno de estos puntos. Gracias a esta característica no necesitamos recurrir a algoritmos como Viola-Jones o transformada de Haar ya la detección de los puntos se realiza internamente en la Kinect, y dado que la posición de los puntos se actualiza con cada *frame* no necesitamos emplear KLT para el análisis de las trayectorias de los POI. Del mismo modo que se hace en el algoritmo base se realiza un filtro Butterworth que filtra las frecuencias que recorta las frecuencias en el rango de 0.5Hz a 4 Hz que se correspondería con 30 y

240 pulsaciones respectivamente. En esta adaptación del algoritmo se ha prescindido del análisis de componentes principales, ya que se consideró que el número de variables es muy pequeño de por sí, solo cuenta con 5 valores por *frame*. Y finalmente se aplica la transformada rápida de Fourier (fft) con una longitud de 1024 puntos cuyo máximo nos proporciona el valor de las pulsaciones. Este algoritmo está desarrollado e la función *calculateHeartRate.* Además, para no limitar la cantidad de resultados en caso de emplear un número de medidas muy alto se ha desarrollado un mecanismo de ventana deslizante que permite aplicar el algoritmo cada 64 mediciones independientemente del tamaño de la muestra.

En este caso se ha eliminado los análisis basados en imágenes de profundidad, ya que como se podía comprobar en el trabajo de Guillermo Guillen [17] la cámara de profundidad del Kinect v2 posee una serie de franjas en las que se pierde una gran cantidad de detalle en las imágenes, y no se podrían aplicar estos algoritmos.

4.4 Controlador

El controlador es el elemento que se encarga de regular la transmisión de información entre la vista y el modelo.

En este caso el controlador se comporta como una API REST. Al desplegar la aplicación se le asigna un puerto aleatorio entre el conjunto de puertos disponibles del sistema. Una vez desplegada cada petición POST realizada contra este puerto será procesada por el controlador; sin embargo, solo los parámetros esperados son procesados. Tras realizar el análisis de estos parámetros se realizan las actuaciones correspondientes sobre los módulos.

En el caso de indicar una tasa de refresco distinta a "*source*" desde la UI el controlador se encarga de realizar las peticiones al Kinect para acceder al último *frame* almacenado como 1000/tasa_refresco milisegundos, y aplicando toda la lógica descrita en el módulo Kinect.

En caso de realizar una grabación el controlador genera un mecanismo para el almacenamiento en disco de *frames* en color, *frames* en profundidad, ritmo cardíaco real, ritmo cardíaco estimado y puntos de interés empleados para dicha estimación. Para no interferir con el funcionamiento de la aplicación y no alterar la tasa de refresco se emplean dos colas de acceso concurrente (**ConcurrentQueue)** que almacenan los *frames* de color y profundidad para ser procesados paralelamente por un proceso en un

hilo. De esta manera los *frames* se van procesando de uno en uno en un hilo y no se reduce el *frame* rate.

Sin embargo, por el hecho de almacenar estas imágenes en colas hace que durante ese periodo el proceso consuma más memoria RAM de lo normal y por esto se ha limitado a 20 segundos para evitar excepciones de falta de memoria

4.5 Vista

La vista es la encargada de mostrar en el navegador el estado actual de la aplicación. Para el desarrollo de las vistas se han empleado HTML, CSS, Javascript, JQuery y Razor.

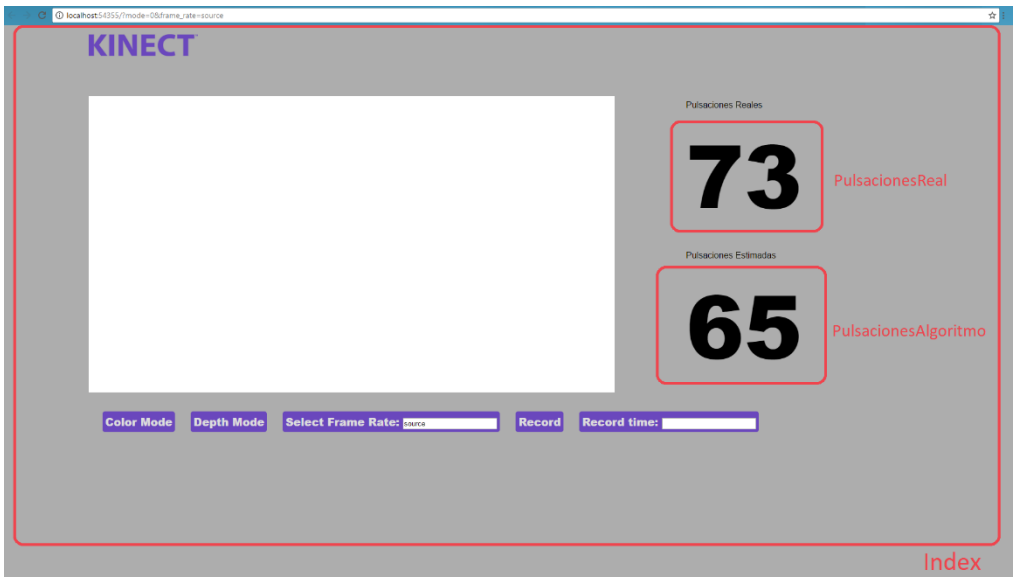

Figura 4.7: Vistas parciales de la Interfaz de Usuario

La UI de la aplicación se compone de varias vistas como se puede ver en la Figura 4.7. La vista Index es la vista principal y es la que devuelve el controlador en cada llamada a la aplicación mediante petición post o mediante alguno de los botones de la interfaz. Además, hay dos vistas parciales PulsacionesReal y PulsacionesAlgoritmo que se refrescan en periodos de 1 segundo respectivamente y que solo contiene los resultados obtenidos de los distintos módulos de la aplicación. El uso de vistas parciales permite refrescar solo una sección de la ventana, lo que evita tener que cargas el resto de los recursos como las imágenes.

La imagen que contiene la salida de la Kinect se obtiene leyendo una imagen escrita por el módulo Kinect que se sobrescribe con el mismo nombre y cuyo contenido se actualiza con cada nuevo *frame*. La tasa de refresco de esta imagen es de 50 milisegundos y es independiente de la tasa sacada por el Kinect empleada para la grabación, y solo si la tasa de refresco seleccionada es inferior a 20 *frames*/segundo se podrá apreciar en la UI.

Tanto el refresco de la imagen como de las vistas parciales se realiza médiate funciones javascript: *refreshPulsacionesAlgoritmo(), refreshPulsacionesReal() y refreshKinectImage().*

Para las peticiones POST realizadas al pulsar los botones de **Color Mode**, **Depth Mode** o **Record** se ha empleado JQuery y para la comunicación con el controlador se ha empleado Razor. Es importante destacar que todo el código C# insertado en HTML mediante Razor se genera en el servidor antes de mandar la vista, por lo que no es posible el uso de C# para realizar las lecturas de ficheros que sustituirían a las vistas parciales.

5. Pruebas y Resultados

5.1. Introducción

Finalmente se han realizado una serie de mediciones en varios sujetos con el fin de evaluar cómo funciona el algoritmo implementado sobre la aplicación desarrollada. Una vez presentados los resultados se analizarán para extraer conclusiones sobre los datos.

5.2. Pruebas y resultados

Para probar el funcionamiento del algoritmo se han realizado mediciones del ritmo cardíaco real y del ritmo estimado mediante el algoritmo empleando la aplicación desarrollada. Para evaluar distintos comportamientos se han ido aplicando distintos tamaños de muestra (cantidad de muestras tomadas de un Punto de Interés), distintos tamaños de ventana (cantidad de nuevas muestras que se añaden por segundo) y se comprobado el funcionamiento con y sin el filtrado.

Todas las mediciones se han tomado a una distancia de 2.5 metros del Kinect, ya que ha distancias menores el sensor tiene dificultades para identificar cuerpos y por lo tanto no permite acceder a la información de los puntos de interés.

Para las pruebas se han prescindido de los puntos de interés de los ojos, ya que producían problemas en el muestreo y daban valores muy altos. Si se han empleado los 3 puntos restantes, los puntos de la boca (esquina izquierda y esquina derecha) y la nariz (punta).

En cada una de las sesiones se monitorizaba entre 30 y 60 segundos de vídeo dependiendo del tamaño de la muestra, ya que si el tamaño de la muestra es de 1024 medidas necesitamos grabar 40 segundos para obtener el primer resultado más n segundos dependiendo del número de resultados que queremos medir, en nuestro caso se emplearon 25 muestras por tamaño de ventana. Cada segundo se almacenaban los valores reales y estimados del ritmo cardíaco empleando como resultado final la media de estas 25 muestras.

Para el cálculo del error se ha empleado la siguiente formula:

$$\%\text{Error} = \left|\frac{\text{Media pulso real} - \textit{Media pulso estimado}}{\textit{Media real}}\right| 100$$

Se han comparado tamaño de muestras de 128, 256, 512 y 1024 valores ya que se ha comprobado que el número de imágenes empleados para el cálculo repercute notablemente en la variabilidad del resultado como se puede apreciar en la figura 5.1.

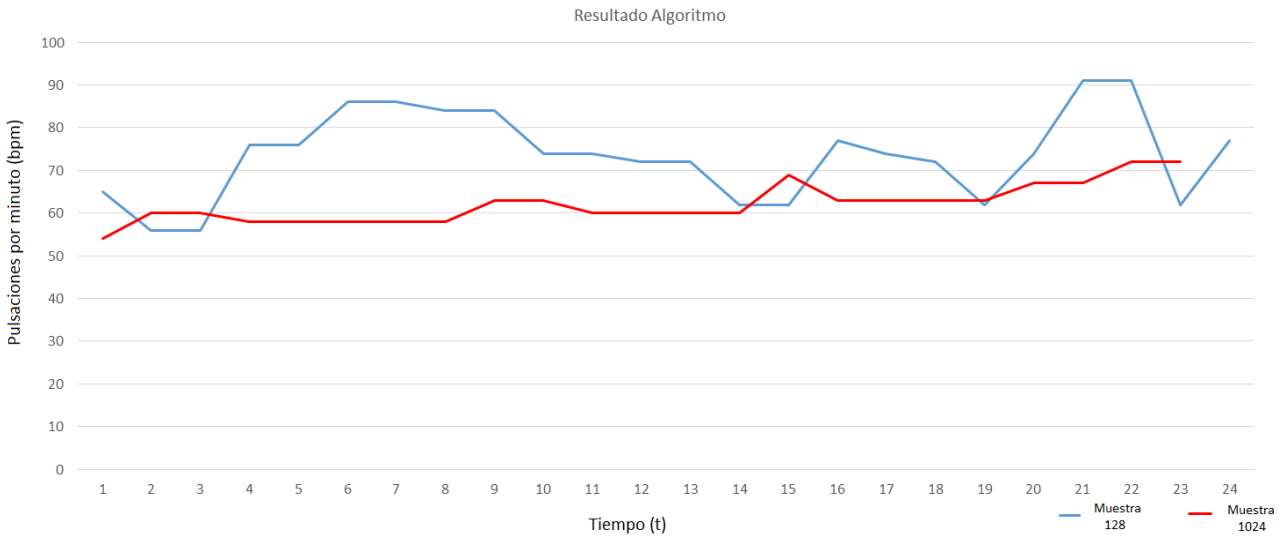

Figura 5.1: Resultados del algoritmo con muestras de 128 y 1024 muestras.

A continuación, se exponen los resultados de las mediciones realizadas. En primer lugar aplicando el filtro Butterworth sobre las diferencias de posición antes de aplicar la fft. En segundo lugar, se muestran los resultados sin aplicar el filtrado

Tamaño de la muestra	Tamaño de la ventana	Valor Real	Valor estimado Algoritmo	%Error
128	32	74,08	57,00	23,06
	64	76,67	48,00	37,39
256	32	71,71	60,58	15,52
	64	80,38	70,85	11,85
512	32	67,67	64,01	5,40
	64	75,04	59,01	21,36
1024	32	72,21	62,55	13,38
	64	70,75	61,25	13,42

Tabla 5.2: Resultado del algoritmo empleando filtrado.

Tamaño de la muestra	Tamaño de la ventana	Valor Real	Valor estimado Algoritmo	%Error
128	32	74,33	61,37	17,44
	64	79,22	58,20	26,53
256	32	70,08	60,75	13,31
	64	77,30	94,49	22,24
512	32	84,41	64,01	24,17
	64	68,74	54,31	20,99
1024	32	74,37	81,25	9,25
	64	71,24	59,96	15,83

Tabla 5.3: Resultado del algoritmo sin emplear filtrado.

Finalmente se puede verificar mediante las figuras 5.2 y 5.3 que el incremento del tamaño de la muestra, a parte de la estabilidad de los resultados, realiza una mejor aproximación del valor real proporcionado por el pulsímetro.

En la prueba realizada con el tamaño de muestra de 1024 se puede apreciar cómo se produce un mejor seguimiento de las variaciones producidas en el valor real del ritmo cardiaco, a diferencia del ejemplo con un tamaño de muestra de 128 donde la fluctuación de los valores impide apreciar este comportamiento.

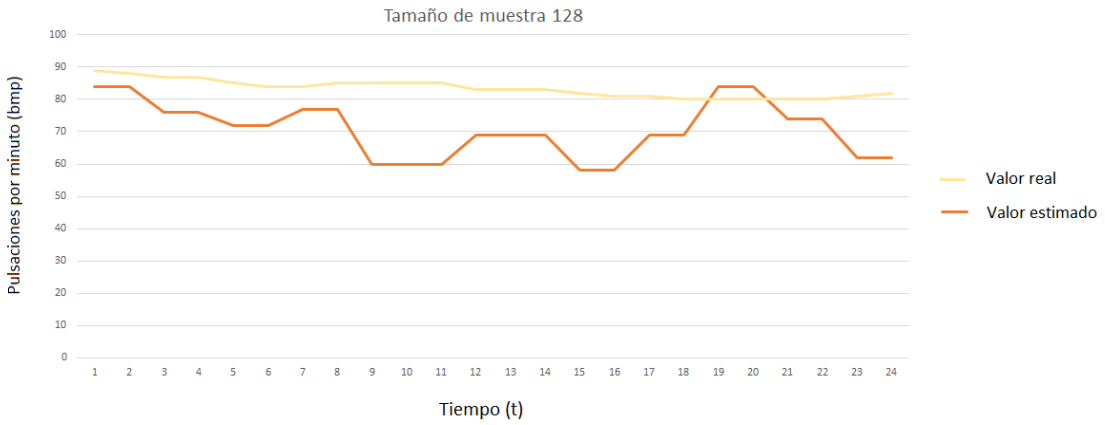

Figura 5.2: Comparación valor estimado – valor real con un tamaño de muestra de 128.

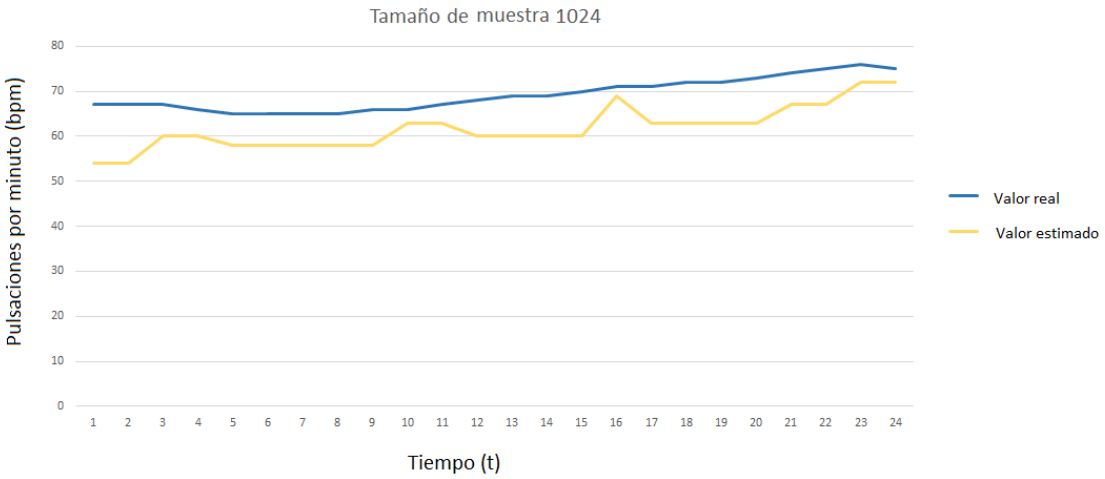

Figura 5.3: Comparación valor estimado – valor real con un tamaño de muestra de 1024.

5.3 Conclusiones

A raíz de los resultados mostrados se deduce que:

- Un tamaño de muestra más grande mejora los resultados del algoritmo, ya que permite identificar más fácilmente el patrón del pulso mediante la fft. Sin embargo, si aumentamos el tamaño de venta a 2048 apenas se aprecia mejora alguna y los resultados son muy similares a los mostrados con 1024.
- Cuanto mayor es el ritmo cardíaco del usuario peor es el resultado del algoritmo. Los valores mostrados en las pruebas eran tomadas sobre sujetos en reposo. Al

realizar las pruebas sobre usuarios con las pulsaciones sobre 120 el error del algoritmo subía notablemente.

- El filtrado empleado sobre las diferencias de posición de los puntos de interés mejora el resultado del algoritmo, ya que reducen las posibles variaciones resultado de movimientos involuntarios.

En comparación con los resultados obtenidos por Erik en [16] se puede apreciar un empeoramiento de los resultados (entorno al 3%), debido a que los resultados del algoritmo desarrollado debían generarse en intervalos de tiempo más pequeños.

6. Conclusiones y trabajo futuro

6.1 Conclusiones

El objetivo de este trabajo de fin de grado era el de desarrollar una aplicación que integrase un algoritmo de análisis de secuencia de vídeo con un mecanismo que permitiese comprobar las pulsaciones reales del usuario y todo ello en una aplicación que facilitase el acceso al resultado de ambos y que permitiese su grabación para su uso en la creación de *datasets* con *ground-truth* que sirvan para diseñar y evaluar otros algoritmos de estimación de ritmo cardíaco a partir de secuencias de video. Tras analizar las diferentes tecnologías existentes dentro del escenario de Windows, Visual Studio, y C# se han desarrollado una serie de módulos siempre tratando de crear un desarrollo claro que facilitase la edición del código en futuros trabajos.

Se han desarrollado módulos que permiten interactuar con la Kinect 2, acceder a características de dispositivos BLE, aplicar algoritmos de tratamiento de imagen en tiempo real almacenando los resultados y modificando propiedades como el tipo de imagen o la tasa de refresco de las imágenes empleadas.

A pesar no mejorar los resultados del algoritmo realizado en Matlab en [16] la aplicación creada permite un desarrollo más rápido de futuros algoritmos desarrollados en lenguajes de alto nivel como C#.

6.2 Trabajo futuro

A raíz de los resultados obtenidos se propone trabajar en:

- Emplear las características de *Kinect Face HD* en sustitución de *Kienct Face*. Esta característica permitiría seleccionar entre 1400 puntos de interés identificados en

el rostro del usuario además de mejorar la precisión a la hora de analizar el movimiento de estos. Empleando esta característica se podría comprobar como diferentes zonas de la cara mejoran o empeoran el resultado del algoritmo.

- Adaptar la implementación a la versión 2017 de Visual Studio y C# 7, lo que mejoraría el rendimiento general de la aplicación y permitiría reducir el *footprint* de la aplicación.

- Implementar el algoritmo de análisis de ritmo cardíaco sobre imágenes Infrarrojas. Las imágenes en profundidad se descartaron por no mejorar los resultados comparados con las imágenes en color, pero un algoritmo basados en imágenes infrarrojas permitiría abstrae los resultados de las situaciones lumínicas del escenario y también estar integrado con las herramientas de *Kinect Face*.

- Implementar las funciones de acceso y manejo del Kinect en C++ con el uso de librerías de OpenCV con el fin de eliminar la dependencia con Visual Studio.

Referencias

[1] H.-Y. Wu, M. Rubinstein, E. Shih, J. Guttag, F. Durand, and W. Freeman, “Eulerian video magnification for revealing subtle changes in the world,” ACM Transactions on Graphics, vol. 31, no. 4, pp. 1-8, 2012.

[2] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features.” Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), vol. 1, pp. 511-518, 2001.

[3] A. Kleiner and D. Rabinowitz, Video Heart Rate Detection, 2014

[4] M. Z. Poh, D. J. McDu, and R. W. Picard, Advancements in noncontact, multiparameter physiological measurements using a webcam, IEEE Transactions on Biomedical Engineering, vol. 58, no. 1, pp. 7-11, 2011.

[5] G. R. Tsouri, S. Kyal, S. Dianat, and L. K. Mestha, Constrained independent component analysis approach to nonobtrusive pulse rate measurements, Journal of biomedical optics, vol. 17, no. 7, pp. 0770111-0770114, 2012.

[6] T. Fitzpatrick, Sun and skin (soleil et peau), J Med Esthétique, vol. 2, pp. 33-34, 1975.

[7] F. Bousefsaf, C. Maaoui, and A. Pruski, Continuous wavelet ltering on webcam photoplethysmographic signals to remotely assess the instantaneous heart rate, Biomedical Signal Processing and Control, vol. 8, pp. 568-574, Nov. 2013.

[7] C. Takano and Y. Ohta, Heart rate measurement based on a time-lapse image., Medical engineering & physics, vol. 29, pp. 853-857, Oct. 2007.

[9] L. Carvalho, M. Virani, and M. Kutty, Analysis of Heart Rate Monitoring Using a Webcam, Analysis, vol. 3, no. 5, pp. 6593-6595, 2014.

[10] S. Kwon, H. Kim, and K. S. Park, Validation of heart rate extraction using video imaging on a built-in camera system of a smartphone, Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS, pp. 2174-2177, 2012.

[11] J. B. Bolkhovsky, C. G. Scully, and K. H. Chon, Statistical analysis of heart rate and heart rate variability monitoring through the use of smart phone cameras, in Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the IEEE, pp. 1610-1613, IEEE, 2012.

[12] G. Balakrishnan, F. Durand, and J. Guttag, “Detecting pulse from head motions in video,” Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 3430-3437, June 2013.

[13] J. Shi and C. Tomasi, “Good features to track,” in Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on, pp. 593-600, IEEE, 1994.

[14] B. D. Lucas, T. Kanade, et al., “An iterative image registration technique with an application to stereo vision.,” in IJCAI, vol. 81, pp. 674-679, 1981.

[15] R. Irani, K. Nasrollahi, and T. B. Moeslund, “Improved Pulse Detection from Head Motions Using DCT,” International Conference on Computer Vision Theory and Applications, p. 8, 2014.

[16] Erik Velasco Salido, Detección de ritmo cardíaco mediante vídeo, Trabajo Fin de Grado, Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación, Escuela Politécnica Superior, Universidad Autónoma de Madrid, Junio 2015.

[17] Guillermo Guillén Mier, Detección de ritmo cardíaco mediante video-monitorización. Escuela Politécnica Superior, Universidad Autónoma de Madrid, Junio 2016.

[18] https://msdn.microsoft.com/en-us/library/windowspreview.kinect.aspx

[19] https://www.bluetooth.com/specifications/adopted-specifications

[20] https://docs.microsoft.com/en-us/uwp/api/

[21] https://es.mathworks.com/help/matlab/matlab_external/call-matlab-function-from-c-client.html

[22] https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.heart_rate.xml&u=org.bluetooth.service.heart_rate.xml

[23] https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.characteristic.heart_rate_measurement.xml&u=org.bluetooth.characteristic.heart_rate_measurement.xml

[24] https://developer.polar.com/wiki/H6_and_H7_Heart_rate_sensors

[25] https://msdn.microsoft.com/es-es/library/dd381412(v=vs.108).aspx

[27] https://www.cardiio.com/

[28] http://sport.com/heart_rate_monitor/

Apéndice A: Manual de Instalación

Material Necesario

Para el desarrollo completo del funcionamiento de esta aplicación se enumeran a continuación los recursos tanto software como hardware.

Elementos software:

- Se requiere de un ordenador con Windows 8.1 o superior (desarrollado en Windows 10).
- Visual Studio 2015 (no valido con Visual Studio 2017).
- Kinect for Windows SDK 2.0.

Como elementos adicionales (pero no imprescindibles) para testear el correcto funcionamiento del módulo de ritmo cardíaco se ha empleado un ordenador con Ubuntu en el que será necesario tener instalado el paquete blueZ y Samba.

Elementos hardware:

- Banda de frecuencia cardiaca Polar H7
- Kinect 2
- Adaptador de Kinect 2 para PC Windows.

Probado en Google Chrome, las características .css, Ajax, y jQuery no se han probado en los demás navegadores

Despliegue de la aplicación

Acceso al código de la aplicación:

https://www.dropbox.com/sh/t673q8911v4uli7/AACZdrbEf_rRO8UwVjpnuFVla?dl=0

Paquetes necesarios en Ubuntu:

- blueZ se encuentra instalado por defecto en las últimas distribuciones de Linux. Para verificar que no hace falta instalar ningún paquete basta con ejecutar los manuales de hcitool y gatttool.
- Golang solo será necesario en caso de que se necesite recompilar fichero .go en caso de modificación. Instalación: https://golang.org/doc/install.
- Samba también se encuentra instalado por defecto en alguna distribución de Linux. En caso de no disponer de este paquete ejecutar: `sudo apt-get install Samba`

Para verificar que la banda polar se comunica correctamente con los scripts ejecutar en dos terminales diferentes:

- sudo ./heart_rate.sh
- sudo ./heart_rate_file_reader pulsaciones_raw.txt <fichero_de_salida>

Ambas ejecuciones se pueden parar con Ctrl + c. Si la comunicación se ha establecido en el fichero pulsaciones_raw.txt debería haber algo como esto:

```
sergio@sergiogr:~/Escritorio/golang_module$ cat pulsaciones_raw.txt
Characteristic value was written successfully
Notification handle = 0x0012 value: 06 56
Notification handle = 0x0012 value: 16 56 d4 02
Notification handle = 0x0012 value: 16 57 b5 02 8d 02
Notification handle = 0x0012 value: 16 59 6f 02 56 02
Notification handle = 0x0012 value: 16 5a 64 02
Notification handle = 0x0012 value: 16 5b 7a 02 93 02
Notification handle = 0x0012 value: 16 5c 85 02
Notification handle = 0x0012 value: 16 5c 96 02 8c 02
```

Y en el fichero de salida

```
12:13:25, value: 73
```

Para desplegar la aplicación es necesario creo una carpeta compartida entre el ordenador Windows que correrá la aplicación y un ordenador Ubuntu con los paquetes mencionados anteriormente. Para compartir una carpeta entre los sistemas:

- Ubuntu: Click der. sobre la carpeta que contiene <fichero_de_salida>. En la opción “Recurso compartido de red local” marcamos “Compartir esta carpeta”, “Permitir a otras personas crear y eliminar archivos en esta carpeta” y “Acceso invitado”. Y en la siguiente pestaña seleccionamos “Añadir los permisos automáticamente”

Si la carpeta se ha compartido correctamente bastara con ejecutar:

- Windows: Windows+r y en el formulario escribir \\<ip del ordenador ubuntu>\<nombre de la carpeta compartida>

Y se podrá ver desde Windows el contenido de la carpeta.

Finalmente modificar en el fichero /Kinect_Heart_Rate/Controller/HomeController.cs la variable `sharedHeartRateFile` con el valor empleado en los pasos anteriores.

Apéndice B: Kinect Face y Kinect Face HD

Kinect Face y *Kinect Face HD* son dos librerías que permiten emplear la Kinect como elemento detector de caras, en este trabajo se ha empleado la versión simplificada ya que se estimó que sería suficiente para el caso de uso de la aplicación. A continuación, se exponen las principales características y diferencias de estas herramientas.

Kinect Face

Esta versión provee un acceso más simplificado y sencillo. Esta versión nos permite acceder a un máximo de 5 puntos detectados en la cara del usuario, proporcionados como un *array* de **PointF**. Estos puntos son:

- Esquina izquierda de ojo izquierdo
- Esquina derecha de ojo derecho
- Esquina izquierda y derecha de la boca
- Punta de la nariz

Estos puntos se obtienen a partir de imágenes a color o de las imágenes infrarrojas de pendiendo de los canales que se hayan abierto durante la inicialización del **FaceFrameSource**.

```
            faceSource = new FaceFrameSource(sensor, trackedBodyId,
FaceFrameFeatures.PointsInColorSpace, FaceFrameFeatures.PointsInInfraredSpace);
```

Y una vez realizado la configuración del **FaceFrameSource** podremos acceder a la posición de cada uno de estos puntos en cada *frame* del tipo **FaceFrame** que recibamos.

Además, permite identificar una serie de expresiones sencillas inferidas de los puntos anteriores como son:

- Apertura o cierre de los ojos
- Apertura de la boca
- Alegría

Kinect Face HD

La principal ventaja de esta versión es la precisión aumentada sobre los puntos de interés detectados, a diferencia de la versión anterior esta proporciona más de 1000 puntos con un funcionamiento similar a la versión de Kinect Face.

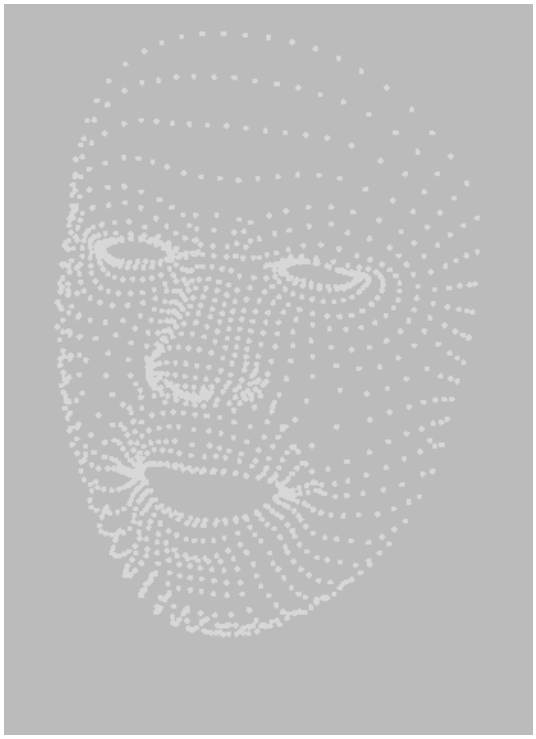

Figura B.1: Puntos proporcionados por Kinecct Face HD

Ambos paquetes se pueden instalar mediante paquetes *nuget*:

Versión de 32 bits:

- `Install-Package Microsoft.Kinect.Face -Version 2.0.1410.19000`

Versión de 64 bits:

- `Install-Package Microsoft.Kinect.Face.x64 -Version 2.0.1410.19000`